(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平6－348595

(43)公開日　平成６年(1994)12月22日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　12/08	３１０　Ｚ	7608－5Ｂ		

審査請求　未請求　請求項の数５　ＦＤ　（全　９　頁）

(21)出願番号　特願平5－163148

(22)出願日　平成５年(1993)６月７日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　久島　伊知郎
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(72)発明者　海永　正博
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(74)代理人　弁理士　笹岡　茂　（外１名）

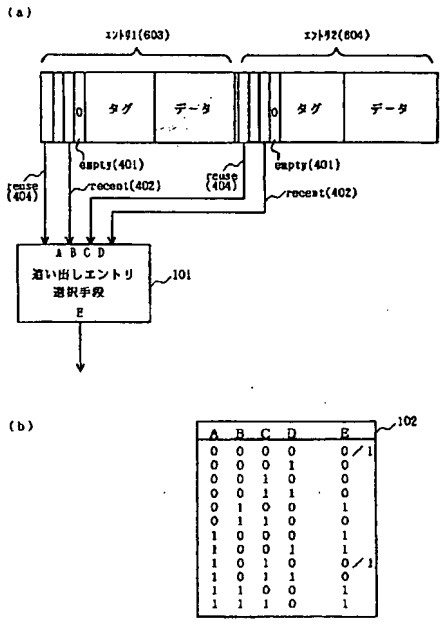
(54)【発明の名称】　キャッシュ装置

(57)【要約】

【目的】　特定のデータを優先的にキャッシュに残しておくことを可能とするキャッシュ装置を提供する。

【構成】　キャッシュの各エントリにデータの追い出しを優先的に阻止するｒｅｕｓｅビットを設け、図１の場合、２セットのエントリのうち一方だけのｒｅｕｓｅビット（４０４）が真になっている場合は、他方のエントリを追い出しエントリとして選択する。両セットのｒｅｕｓｅビットとも真、または両セットとも偽になっている場合は、ｒｅｃｅｎｔビット４０２が偽になっているエントリを選択する。ロード命令の命令語にはＲＥＵＳＥ指定があるかどうかを示すビットを設け、このビットが真であれば、当該ロード命令でアクセスしたデータを含むキャッシュエントリのｒｅｕｓｅビット（４０４）は真となる。

【図１】



1

【特許請求の範囲】

【請求項１】　中央処理装置に接続され、それぞれがデータ領域および制御情報領域を有する複数のエントリを備えるキャッシュ装置において、

エントリ中のデータのキャッシュからの追い出しを優先的に阻止するｒｅｕｓｅ情報を前記制御情報の１つとして設け、

キャッシュ置換時に前記制御情報に基づきキャッシュから追い出すエントリを選択しかつ前記ｒｅｕｓｅ情報の設定されていないエントリを優先して選択する選択手段を具備したことを特徴とするキャッシュ装置。

【請求項２】　請求項１記載のキャッシュ装置において、前記中央処理装置におけるメモリデータの参照と前記ｒｅｕｓｅ情報の値の設定をする命令の実行に応じて、前記エントリにデータを格納すると共に前記ｒｅｕｓｅ情報の値を設定する手段を具備したことを特徴とするキャッシュ装置。

【請求項３】　請求項１または請求項２記載のキャッシュ装置において、前記キャッシュ装置をセットアソシアティブ方式としたことを特徴とするキャッシュ装置。

【請求項４】　請求項１または請求項２記載のキャッシュ装置において、前記キャッシュ装置を２ウェイ・セットアソシアティブ方式としたことを特徴とするキャッシュ装置。

【請求項５】　請求項１または請求項２記載のキャッシュ装置において、前記ｒｅｕｓｅ情報を１ビット情報としたことを特徴とするキャッシュ装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明はキャッシュ装置に係り、特に重複使用されるデータを優先的にキャッシュに残すのに好適なキャッシュ装置に関する。

【０００２】

【従来の技術】キャッシュは、高速に動作するＣＰＵとアクセス速度の遅い主メモリ（以降単にメモリと呼ぶ）の速度のギャップを埋めるために設けられたアクセス速度の速い小容量の記憶装置である。ＣＰＵ、キャッシュとメモリの関係を簡単に示すと図２のようになる。プログラムを実行する際、ＣＰＵが参照するメモリ内容は、もしキャッシュ上に存在していればそれを使い、なければメモリまでアクセスしに行く。キャッシュはＣＰＵに非常に近く、しかも高速なので、ＣＰＵが必要とする多くのデータがキャッシュ上にあれば、非常に高速な処理が期待できる。キャッシュが有効に働くには、アクセス頻度の高いアドレスのデータをキャッシュに置いておくことが必要である。一般的なプログラムのメモリアクセスのパターンを解析すると、次のような特徴が現れる場合が非常に多い。

１．あるのアドレスのデータに対するアクセスは比較的短い時間内に再発する。

2

２．ある一定時間内にアクセスされるデータは比較的近いアドレスに分布する。

前者は「時間的局所性」、後者は「空間的局所性」と呼ばれる。

【０００３】前者の時間的局所性が表すところは「一度アクセスされたアドレスは近い将来再びアクセスされる」ということである。すなわち、今アクセスされたデータをキャッシュにとっておけば、次回そのアドレスが再び参照されたときにはそのデータはキャッシュにあるためアクセス時間が短くて済む。したがって、通常のキャッシュでは、ＣＰＵが参照するデータがキャッシュ上にない場合はそれを必ずキャッシュにもってきて、将来の再参照に備えるようにしている。一方、キャッシュは後者の空間的局所性も利用している。空間的局所性の意味するところは「あるアドレスがアクセスされたら、近い将来その近くのアドレスもアクセスされる」ということである。すなわち、データを新たにメモリからキャッシュに持ってくるときは、今アクセスされたアドレスのデータだけをキャッシュに持ってくるのではなく、その付近のデータも一緒にキャッシュに持ってくるようにすれば、将来その付近のアドレスのデータへのアクセスがあってもそのデータはキャッシュにあるためアクセス時間が短くて済むことが多い。このため、キャッシュは一定長（十数バイトから百バイト程度）のラインごとに分割されており、キャッシュとメモリとのデータのやりとりはライン単位に行われている。例えば、ある１つのデータを読むためにはそれを含む１ラインのデータ全てをキャッシュに持ってくるようにしている。

【０００４】キャッシュはメモリアクセス時間を短縮するのに有効であるが、局所性がないプログラムにに対しては無駄になることがある。例えば次のプログラムを考えてみる。なお、ｆｌｏａｔは浮動小数点データの宣言を意味し、ｉ＋＋はｉに＋１することを意味する。

```
float  a〔1000〕;
for  (i＝0；i＜100；i++) {
    …a〔i＊10〕…
}
```

ここで配列ａへの参照を考える。このプログラムで参照される配列ａの要素は、ａ〔０〕，ａ〔１０〕，ａ〔２０〕，…，ａ〔９９０〕である。それぞれ必ず違った要素が参照されるので、時間的局所性は存在しない。また空間的局所性を考えると、例えばキャッシュの１ラインが３２バイト、配列要素１個が４バイトという想定のもとでは、１ラインには配列の要素は８個分しか保持されず、上のプログラムのように１０個おきにデータがアクセスされる場合にはライン中の他のデータは全く利用されない。すなわち（空間の大きさを３２バイトとすれば）空間的局所性もない。

【０００５】このようなキャッシュへの無駄なロードを避けるため、キャッシュバイパスという機能（命令）を

3

もつプロセッサがある。これはメモリ中のデータを参照する命令について、そのデータがキャッシュ中に存在しない場合でも、それをキャッシュに置かないで直接使用することを指示できるようにしたものである。上のプログラムの配列aへの参照のようにキャッシュにデータを置いても無駄なことがわかっている場合は、その命令を使い、aの要素をキャッシュに置かないようにすることができる。このような機能を備えたプロセッサについては、例えば、リー：「オン・ザ・フローティング・ポイント・パフォーマンス・オブ・ザ・i860・マイクロプロセッサ」、インターナショナル・ジャーナル・オブ・ハイ・スピード・コンピューティング、ヴォリューム4、ナンバー4（1992）、第251頁から267頁（K. Lee: "On the Floating *

```
float a〔100〕, b〔1000〕〔100〕;
for (i=0; i<1000; i++) {
    for (j=0; j<100; j++) {
        …a〔j〕…
        …b〔i〕〔j〕…
    }
}
```

このプログラムは2重ループで、aとbの2つの配列を参照している。aは1次元配列で大きさは100＊4＝400バイト、bは2次元配列で大きさは1000＊100＊4＝400キロバイトである。キャッシュのサイズが32キロバイトという想定であれば、これらの全てのデータをキャッシュに収めることは不可能である。配列aは内側のループではa〔0〕, a〔1〕, a〔2〕, …, a〔99〕と100個の要素が参照される。外側のループによりこれが1000回繰り返され、同じ要素が1000回参照されることになる。すなわち配列aの参照は時間的局所性と空間的局所性（連続するアドレスを順次アクセスするので）を併せ持つ可能性が高い。配列aの大きさはキャッシュサイズより小さいので、全体がキャッシュに収まる。したがって1度配列要素をキャッシュに持ってくれば残りの999回の参照は、理想的にはキャッシュヒット（データがキャッシュ上に存在）するはずである。一方、配列bはループ全体で順にb〔0〕〔0〕, b〔0〕〔1〕, …, b〔0〕〔99〕, b〔1〕〔0〕, b〔1〕〔1〕, …, b〔1〕〔99〕, b〔2〕〔0〕, b〔2〕〔1〕, ……, b〔999〕〔99〕と1000＊100＝100000個の要素が参照される。同じ要素が2度参照されることはないので時間的局所性はないが、空間的局所性はあるのでキャッシュが有効に働く。すなわち、キャッシュの1ラインに配列要素が8個分はいるという想定では、1つの要素（例えばb〔0〕〔0〕）を参照すると後続する7つの要素（b〔0〕〔1〕, …, b〔0〕〔7〕）も一緒にキャッシュにはいるので、後の7回の参照はキャッシュヒットになる。

4

＊Point Preformance of the i860 Microprocessor", International Journal of High Speed Computing, Vol. 4, No. 4 (1992), pp. 251－267）に記載がある。また一般にキャッシュ技術に対する解説としては、情報処理、ボリューム33、ナンバー11（1992）、第1348頁から1357頁に記載がある。

【0006】

【発明が解決しようとする課題】上記従来技術では、空間的局所性と時間的局所性を持つデータを扱うプログラムでキャッシュが有効利用できない場合がある。これを以下のプログラムで説明する。

【0007】さて、配列bの大きさはキャッシュサイズよりも大きいので、その全体をキャッシュにいれることはできない。とするといつかは、以前キャッシュに入っていたデータを追い出して新たにそこにデータを入れるということが必要になる。このとき、どのデータを追い出すかというのが問題になるが、上のプログラムの場合、配列aのデータを追い出すよりも、配列bのデータを追い出したほうがよい。なぜなら配列aの各要素は何度も参照されるけれども配列bのデータは1度しか参照されないからである。一般のキャッシュでは、どのデータをキャッシュから追い出すかはランダムまたはLRU（Least Recently Used）という選択方法を採っている。ランダムは追い出すエントリを文字通りランダムに選ぶもので、データの使用状況は全く考慮されない。LRUは最後にアクセスされた時刻が最も古いデータをキャッシュから追い出すという選択方法である。これは、言ってみればデータの過去の参照状況をみてそれをキャッシュに残すかどうか追い出すかを決めるものである。いずれにしても、そのデータが将来再使用されるかどうかを直接考慮したものではない。従って上のプログラムの場合、配列bが追い出されて配列aが残されるという保証はなかった。すなわち従来技術では、特定のデータ（例えば将来再使用されることが明らかデータ）をキャッシュから追い出さないようにする、ということができなかった。また、プログラマまたはコンパイラが、特定のデータに対し、そのデータが再使用される可能性が高いのでそれをキャッシュから追い出さないようにするという指示を与えることができなかった。この結果、プログラムによってはキャッシュが有効

利用されない（キャッシュヒットが少ない）場合があるという問題点があった。本発明の目的は、特定のデータをキャッシュからなるべく追い出さないようにすることが可能なキャッシュ装置を提供することにある。本発明の別の目的は、プログラマまたはコンパイラが、特定のデータに対し、そのデータをなるべくキャッシュから追い出さないようにするという指示を与えることができる、キャッシュ装置を含む情報処理装置を提供することにある。

【０００８】

【課題を解決するための手段】上記目的を達成するため、中央処理装置に接続され、それぞれがデータ領域および制御情報領域を有する複数のエントリを備えるキャッシュ装置において、エントリ中のデータのキャッシュからの追い出しを優先的に阻止するｒｅｕｓｅ情報を前記制御情報の１つとして設け、キャッシュ置換時に制御情報に基づきキャッシュから追い出すエントリを選択しかつｒｅｕｓｅ情報の設定されていないエントリを優先して選択する選択手段を設けている。また、中央処理装置におけるメモリデータの参照とｒｅｕｓｅ情報の値の設定をする命令の実行に応じて、エントリにデータを格納すると共にｒｅｕｓｅ情報の値を設定する手段を設けている。さらに、キャッシュ装置をセットアソシアティブ方式のキャッシュ装置としている。さらにまた、キャッシュ装置を２ウェイ・セットアソシアティブ方式のキャッシュ装置としている。さらにまた、前記ｒｅｕｓｅ情報を１ビット情報としている。

【０００９】

【作用】課題で挙げたプログラムを例にとって作用の説明をする。プログラムはループの中でａとｂの２つの配列要素の参照があるが、そのうちａの要素（データ）を参照する（ロードする）命令では命令語で、該データをなるべくキャッシュから追い出さないように指示し、ｂの要素を参照する命令では、そのような指示をつけない。これはプログラマまたはコンパイラより行なわれる。メモリアクセス命令は命令語中にその情報を指定するビットを持つようにしているので、これが可能である。データをメモリからキャッシュにフェッチする際、命令語に上記指定があれば、フェッチしたデータを格納するエントリに、当データはなるべくキャッシュから追い出されないようにする、という情報（ｒｅｕｓｅ情報）を付け加える。よってａのデータを格納するキャッシュエントリにはそのような情報が設定され、ｂの要素を格納するエントリには設定されない。以前或るキャッシュエントリに入っていたデータを追い出して新たにそこに別のデータをいれるということが必要になった場合、上記情報が設定されていないエントリが優先的にキャッシュから追い出される。例の場合、ａのデータを格納したエントリよりもｂのデータを格納したエントリからデータが追い出される。このように、特定のデータをキャッシュからなるべく追い出さないようにすることが可能なキャッシュ装置が得られる。また、プログラマまたはコンパイラが、特定のデータに対し、そのデータをなるべくキャッシュから追い出さないようにするという指示を与えることができる、キャッシュ装置を含む情報処理装置が得られる。

【００１０】

【実施例】以下本発明の一実施例を説明する。図２は本発明のキャッシュを含む情報処理装置の構成図である。キャッシュ２０２はアクセス速度の速い小容量の記憶装置で、主メモリ２０４（以降単にメモリ）はアクセス速度の遅い大容量の記憶装置である。キャッシュとメモリはともにバス２０３とつながっており、またＣＰＵ２０１はキャッシュを介してバスとつながっている。プログラムを実行する際、ＣＰＵが参照するメモリ内容は、もしキャッシュ上に存在していればそれを使い、なければメモリまでアクセスしにいく。なお、図２の構成図は従来のキャッシュでも同じである。

【００１１】図３は本発明の一実施例のキャッシュの構成図である。本キャッシュは２ウェイセットアソシアティブ（セット１（３０１）とセット２（３０２）の２つのセットからなるアソシアティブキャッシュ）である。キャッシュ容量は全体で８ｋバイト、すなわち１つのセットの容量は４ｋバイトであり、ここにデータがラインを単位に配列的に格納されている。１セットは０番から１２７番までの１２８個のライン（エントリ）３０６からなる。このラインの番号をインデクス３０７と呼ぶ。１ラインには３２バイトのメモリデータが格納される（３２＊１２８＝４０９６バイト）。１語が４バイトであるとすると、１つのエントリには８語格納可能である。１つのラインは制御ビット３０３、タグ３０４、データ３０５の３つの部分からなる。これについては図４で説明する。

【００１２】図４は本発明のキャッシュの１ラインの構成を示した図である。ライン（３０６）は制御情報が格納される制御情報領域とデータが格納されるデータ領域を有し、制御情報としてｅｍｐｔｙ情報、ｒｅｃｅｎｔ情報、ｄｉｒｔｙ情報、およびｒｅｕｓｅ情報が設けられ、実施例では各情報は１ビット情報とされ、制御情報領域は制御ビット（３０３）からなる。また、データ領域は、実施例ではタグ（３０４）、データ（３０５）からなる。そして制御ビットはｅｍｐｔｙビット（４０１）、ｒｅｃｅｎｔビット（４０２）、ｄｉｒｔｙビット（４０３）、およびｒｅｕｓｅビット（４０４）からなる。データはメモリ中のデータの値を格納するフィールドである。タグは今格納しているデータが、どのメモリアドレスに対応するものかを特定するフィールドである。ｅｍｐｔｙビットは当ラインが空きであるかを示す（真のとき空き）。ｒｅｃｅｎｔビットは当セットのラインが、もう一方のセットのラインよりも後にアクセス

されたかどうか、すなわち最新のアクセスであるかを示す（真のとき最新）。ｒｅｃｅｎｔビットは、本キャッシュがＬＲＵ（Ｌｅａｓｔ　Ｒｅｃｅｎｔｌｙ　Ｕｓｅｄ）に基づく置換方法を用いているため必要となる。ｄｉｒｔｙビットは当キャッシュラインに書き込みがあったかを示す（真のとき書き込みあり）。ｒｅｕｓｅビットは、当ラインのデータが将来再び使用される可能性が高いことを示す。このビットは当該メモリアドレスを参照する命令により設定され、キャッシュ置き換えが必要になったときに参照されるが、これについては後で詳しく説明する。

【００１３】図５はメモリアドレスを２進表現したときの構成を示す図である。アドレスは３２ビットで、タグ５０１、インデクス５０２、ポジション５０３の３つの部分にわかれる。タグは２０ビットでこれがキャッシュラインのタグ（３０４）と対応する。インデクスは７ビットでこれをキャッシュラインのインデクスとして用いる（２の７乗＝１２８であるからこの７ビットによりキャッシュラインが特定できる）。ポジションは５ビットで、ライン中のバイト位置を特定するのに使われる（２の５乗＝３２であるからこの５ビットでバイト位置が特定できる）。

【００１４】次に図６を用いて、メモリアクセスが生じたときの本発明のキャッシュの振舞いを説明する。さて、メモリアクセスが発生したとする。この時メモリアドレス（６０２）のうちインデクス（５０２）の値をインデクスとしてキャッシュライン配列のエントリを特定する。２ウェイのセットアソシアティブであるので、１つのインデクスに対しエントリは２つ（セット１のエントリ６０３とセット２のエントリ６０４）特定される。次に当該のメモリアドレスのタグ（５０１）の値とエントリ内に格納されたタグの値（６０５および６０６）を比較器６０１で比較する。タグの値が等しいエントリ内にはメモリロケーション内容の写しがキャッシュエントリに格納されていると判断する。すなわち、メモリアドレスのタグ値（５０１）とキャッシュエントリのタグ値（６０５または６０６）が等しければキャッシュヒットとなる。この場合メモリアクセスはキャッシュアクセスに置き換えられ、高速アクセスが可能となる。タグの値が等しいエントリがない場合、キャッシュ内にメモリロケーションの写しが格納されてないと判断し、キャッシュアクセスの代わりに低速なメモリアクセスが必要となる。

【００１５】キャッシュ内にメモリロケーションの写しがない場合はさらに、その写しをキャッシュに格納する。これは以下のように行う。タグの値の等しいエントリがなかった場合は、２つのエントリの制御ビットのｅｍｔｐｙビットを調べる。少なくともどちらか一方のｅｍｐｔｙビットが真であれば、そのキャッシュエントリが空きであるので、メモリから読み出したデータをそこに格納する。いずれのエントリのｅｍｐｔｙビットが偽であれば、２つのうち一方をキャッシュから追い出す。このときどちらを追い出すかの判断にｒｅｕｓｅビットとｒｅｃｅｎｔビットを用いる。これを表で示したのが図１であり、以下これを説明する。図１は２つのエントリ１（６０３）とエントリ２（６０４）のうち、どちらを追い出しエントリとするか、選択する様子を示す図である。エントリ１（６０３）はセット１に、エントリ２（６０４）はセット２に属する。２つのセットのエントリの制御ビットの中のｒｅｕｓｅビットとｒｅｃｅｎｔビットを、追い出しエントリ選択手段１０１の入力とし、追い出すエントリを選択する。選択手段１０１は上記の４つのビットを入力とし、０か１かを出力する。０はエントリ１を１はエントリ２を選択することを意味する。入力と出力の関係を示したのが図１（ｂ）の表１０２である。この表の意味するところを説明すると次のようになる。２セットのうち一方だけにｒｅｕｓｅビットが立っている（真になっている）場合は、そうでない（偽になっている）方のセットのエントリを選択する。両セットのｒｅｕｓｅビットとも真、または両セットとも偽になっている場合は、２セットの内の古い（ｒｅｃｅｎｔビットが偽）エントリを選択する（ｒｅｃｅｎｔビットは初期状態では両セットとも偽となっているが、そのアクセスがあれば必ず一方のセットが真、他方が偽となっていて、両方が真となることはない）。ｒｅｃｅｎｔビットが両方とも偽になっている場合はどちらを選択してもよい（０／１により表示）。追い出すエントリが決まったら、そのエントリ内容が変更されているかどうか（ｄｉｒｔｙビットが真かどうか）調べ、そうならばこのエントリの内容をメモリに書き込む。変更されていなければ（ｄｉｒｔｙビットが偽）何もしない。古いエントリの追い出しが完了すれば、メモリロケーション内のデータを読み出しその値を、追い出したキャッシュエントリに格納し、そのエントリのｒｅｃｅｎｔビットを真にし、追い出されなかった方のエントリのｒｅｃｅｎｔビットを偽とする。

【００１６】図７は本発明で適用するロード命令と、それがキャッシュエントリのｒｅｕｓｅビットに設定される様子を示した図である。ロード命令の命令語７０２にはＲＥＵＳＥ指定があるかどうかを示すビット（７０４）を含む。このビットが真であれば、当該ロード命令でアクセスしたデータを含むキャッシュエントリ（３０６）のｒｅｕｓｅビット（４０１）は真となる。

【００１７】次に図８のプログラムを例にとって、従来のキャッシュシステムと本発明のキャッシュシステムの振舞いの違いを説明するが、その前に従来のキャッシュラインの制御ビットと追い出しエントリの選択方法について簡単に説明する。なお、図８において、ｉｎｔは整数（ｓ）の宣言を意味し、ｔ＋＝２は２を加えることを意味する。従来のキャッシュシステムにおけるキャッシ

ュラインの制御ビットはｅｍｐｔｙビット、ｒｅｃｅｎｔビット、およびｄｉｒｔｙビットの3つのビットから構成されている。そして、キャッシュラインが競合したときは、ｒｅｃｅｎｔビットが偽のエントリ、すなわち最後にアクセスされたのがより古い方のエントリをキャッシュから追い出すようにしている。図8のプログラムの2重ループにおいて、外側ループ（８０２）の1回目のイタレーション（繰返し（単位）、例えば、９９回のループがあるとき、その各回をイタレーションという）が完了したとする。この時点で、配列ａ，ｂのアクセスした部分は次の通りである。

ａ〔0．．1〕〔1．．１０２３〕，ｂ〔1．．１０２３〕

ここで例えばｂ〔1．．１０２３〕はｂ〔1〕，ｂ〔2〕，…，ｂ〔１０２３〕を表す。このうち、配列ａは外側ループのイタレーションの進行に伴いアクセス位置が変化していく。一方配列ｂ（外側ループについては）はアクセス位置が変化しない（というか配列ｂ全体であるが、その全容量はサイズ的にキャッシュに収まる）。したがって配列ｂ〔1．．１０２３〕が長い間キャッシュに保持され、外側ループの個別のイタレーションで再利用できるなら高速化の面で都合がよい。しかし、2ウェイセットアソシアティブキャッシュの場合には配列ｂがずっとキャッシュに保持されるとは限らない。というのは配列ｂの要素ｂ〔ｊ〕は配列ａの要素ａ〔ｉ〕〔ｊ〕、要素ａ〔ｉ＋1〕〔ｊ〕とキャッシュラインを競合するからである。それを以下説明する。なおここで、2次元配列は2次元目（ｊの方）が速く変化するようにメモリ上に配置されるとする。図5で示したように、あるアドレスのデータが入るべきキャッシュライン（インデクス）はその中位7ビットによって決まる。従って2つの配列要素があったときそのアドレスの中位7ビットが等しければそれらは同じラインに入る（キャッシュラインの競合）。ただし2ウェイセットアソシアティブであれば1つのインデクスに2つのエントリが対応する。ｂ〔ｊ〕のアドレスは「配列ｂの先頭アドレス（ｂ〔0〕のアドレスのこと）＋ｊ＊4」、ａ〔ｉ〕〔ｊ〕のアドレスは「配列ａの先頭アドレス（ａ〔0〕〔0〕のアドレスのこと）＋ｉ＊４０９６＋ｊ＊4」である。よってａ〔ｉ〕〔ｊ〕とａ〔ｉ＋1〕〔ｊ〕のアドレスはちょうど４０９６だけ異なり、この2つは明らかに同じキャッシュラインに入る。また、ａの先頭アドレスとｂの先頭アドレスの差が４０９６の倍数であれば（ａの大きさは４０９６バイトなので、これは成り立つ可能性が高い）ａ〔ｉ〕〔ｊ〕とｂ〔ｊ〕もやはり同じキャッシュラインに入る。故に、ｂ〔ｊ〕，ａ〔ｉ〕〔ｊ〕，ａ〔ｉ＋1〕〔ｊ〕はキャッシュラインを競合する。

【００１８】図9は、図8のプログラムを機械語イメージに近付けたプログラムである。機械語イメージに近付けたというのは、いま注目している配列ａ，ｂへのアクセスを明示的にｌｏａｄ（）により示しているということである。ここで、ｒ0、ｒ1、ｒ2はレジスタを指す。ループ内の配列要素アクセスはａ〔ｉ〕〔ｊ〕、ａ〔ｉ＋1〕〔ｊ〕、ｂ〔ｊ〕の順に行われる。内側ループのイタレーションを固定（ｊの値が固定のＪ）し、外側ループのイタレーションを順次に増大させて以下従来のキャッシュの振舞いを考える。さて、外側ループの初回のイタレーション（ｉの値は0）において、3つの配列要素参照ａ〔0〕〔Ｊ〕，ａ〔1〕〔Ｊ〕，ｂ〔Ｊ〕はキャッシュラインを競合し、そして置換方式がＬＲＵであるので、3つのうちａ〔1〕〔Ｊ〕とｂ〔Ｊ〕がキャッシュに残る。このうちｂ〔Ｊ〕のｒｅｃｅｎｔビットが真となる。外側ループの次のイタレーション（ｉの値は2）では、最初ａ〔2〕〔Ｊ〕がアクセスされる。この時点で対応するキャッシュエントリにはａ〔1〕〔Ｊ〕とｂ〔Ｊ〕が残っているが、ａ〔1〕〔Ｊ〕の方のｒｅｃｅｎｔビットが偽であるので、ａ〔1〕〔Ｊ〕がキャッシュから追い出され、ａ〔2〕〔Ｊ〕がキャッシュにはいる。その際追い出されなかった方のｂ〔Ｊ〕のｒｅｃｅｎｔビットが偽になり、新しくキャッシュに入ったａ〔2〕〔Ｊ〕のｒｅｃｅｎｔビットが真になる。次にａ〔3〕〔Ｊ〕がアクセスされ、これに伴いｂ〔Ｊ〕がキャッシュから追い出され、ａ〔3〕〔Ｊ〕がキャッシュにはいる。最後にｂ〔Ｊ〕がアクセスされ、ａ〔2〕〔Ｊ〕がキャッシュから追い出されｂ〔Ｊ〕がキャッシュにはいる。外側ループイタレーションの3回目以降は2回目以降とほぼ同様になる。ここで注目すべきはｂ〔Ｊ〕が一旦追い出され、それから新たにフェッチされている点である。つまり外側ループイタレーションの任意の回（ｉが任意の値、ｊが任意の値Ｊ）で、ｂ〔Ｊ〕のメモリからのフェッチが発生している点である。ｊの値は任意であったのでこれは配列ｂの全体がキャッシュに全く残っていなかったことを意味する。以上、従来のキャッシュシステムではキャッシュをまったく有効に活用できないことを示した。

【００１９】次に、本発明のキャッシュシステムで上記のプログラムがキャッシュを有効利用できることを示す。図8を本発明のシステムの機械語イメージに近付けたプログラムを図１０に示す。このプログラムではメモリアクセスをｌｏａｄ（）で明示指定し、その際ＲＥＵＳＥかどうかを指定してある。さて、本発明のキャッシュで先程と同様に図１０のプログラムの実行を追ってみる。内側ループのイタレーションを固定（ｊの値が固定のＪ）し、外側ループのイタレーションを順次に増大させて以下考える。外側ループの初回のイタレーション（ｉの値は0）において、3つの配列参照ａ〔0〕〔Ｊ〕，ａ〔1〕〔Ｊ〕，ｂ〔Ｊ〕はキャッシュラインを競合し、そして置換方式がＬＲＵであるので、3つのうちａ〔1〕〔Ｊ〕とｂ〔Ｊ〕がキャッシュに残る。そ

してｂ〔Ｊ〕はｒｅｃｅｎｔビットが真となる（ａ〔１〕〔Ｊ〕はｒｅｃｅｎｔビットが偽）。さらにｂ〔Ｊ〕のアクセスはＲＥＵＳＥ指定であり（１００６）そのｒｅｕｓｅビットも真となる。外側ループの次のイタレーション（ｉの値は２）でも３つの配列参照ａ〔２〕〔Ｊ〕、ａ〔３〕〔Ｊ〕、ｂ〔Ｊ〕がある。最初ａ〔２〕〔Ｊ〕がアクセスされることに伴いａ〔１〕〔Ｊ〕がキャッシュから追い出され（ｒｅｕｓｅビットもｒｅｃｅｎｔビットも偽のため）、ａ〔２〕〔Ｊ〕がキャッシュに入る。次にａ〔３〕〔Ｊ〕がアクセスされ、これに伴いａ〔２〕〔Ｊ〕とｂ〔Ｊ〕のどちらかがキャッシュから追い出されることになる。ａ〔２〕〔Ｊ〕はｒｅｕｓｅビットが偽、ｒｅｃｅｎｔビットが真であり、一方ｂ〔Ｊ〕はｒｅｕｓｅビットが真、ｒｅｃｅｎｔビットが偽である。従って図１で示した通りａ〔２〕〔Ｊ〕が追い出されるエントリとして選ばれ、ｂ〔Ｊ〕が残る。そしてａ〔３〕〔Ｊ〕がキャッシュに入る。最後にｂ〔Ｊ〕がアクセスされるが、この場合ｂ〔Ｊ〕がすでにキャッシュにあり（キャッシュヒット）、メモリからロードする必要はない。もちろんｂ〔Ｊ〕はｒｅｃｅｎｔかつｒｅｕｓｅとなる。外側ループイタレーションの３回目以降は２回目とほぼ同様になるが、念のために３回目も示しておく。外側ループの３回目のイタレーション（ｉの値は４）では、ａ〔４〕〔Ｊ〕、ａ〔５〕〔Ｊ〕、ｂ〔Ｊ〕の３つが参照される。最初ａ〔４〕〔Ｊ〕がアクセスされこれに伴いａ〔３〕〔Ｊ〕がキャッシュから追い出されａ〔４〕〔Ｊ〕がキャッシュに入る。次にａ〔５〕〔Ｊ〕がアクセスされこれに伴いａ〔４〕〔Ｊ〕とｂ〔Ｊ〕のうち一方がキャッシュから追い出されることになる。ａ〔４〕〔Ｊ〕はｒｅｃｎｅｔ、ｂ〔Ｊ〕はｒｅｕｓｅであり、したがってａ〔４〕〔Ｊ〕が追い出される。そしてａ〔５〕〔Ｊ〕がキャッシュに入る。最後にｂ〔Ｊ〕がアクセスされこの場合ｂ〔Ｊ〕がすでにキャッシュにあり、キャッシュヒットになる。ここで注目すべきはｂ〔Ｊ〕が一旦キャッシュにフェッチされると以降は一度も追い出されていない点である。つまり外側ループイタレーションの初回で配列ｂ全体がメモリからキャッシュにフェッチされると、以降フェッチが全く発生していない点である。言い換えると、配列ｂへのアクセスは時間的局所性をフルに引き出せたということで、これが本発明のキャッシュの利点である。

【００２０】

【発明の効果】本発明によれば、キャッシュの各エントリが、そのエントリのデータをなるべくキャッシュから追い出さないようにすることを示すｒｅｕｓｅ情報を持つので、キャッシュ置換時、どのエントリをキャッシュから追い出すかを決定する際、この情報を利用できる。すなわち、その情報を持つエントリを優先的にキャッシュに残すようにできる。また、メモリデータの参照とｒｅｕｓｅ情報の値の設定をする命令の実行に応じて、エントリにデータを格納すると共にｒｅｕｓｅ情報の値が設定されるので、ユーザまたはコンパイラが特定のデータをなるべくキャッシュに残すように指示することが可能となる。そして、キャッシュミスが少なくすることが可能となる。

【図面の簡単な説明】

【図１】本発明の一実施例の追い出しエントリの選択に係る構成および選択を説明するテーブルを示す図である。

【図２】キャッシュを含む情報処理装置の構成を示す図である。

【図３】本発明の一実施例のキャッシュの構成を示す図である。

【図４】本発明の一実施例のキャッシュの１ラインの構成を示す図である。

【図５】メモリアドレスの構成を示す図である。

【図６】キャッシュアクセスの仕組みを説明するための図である。

【図７】本発明で適用されるロード命令および該命令のキャッシュに対する作用を示す図である。

【図８】キャッシュの動作を説明するためのプログラムを示す図である。

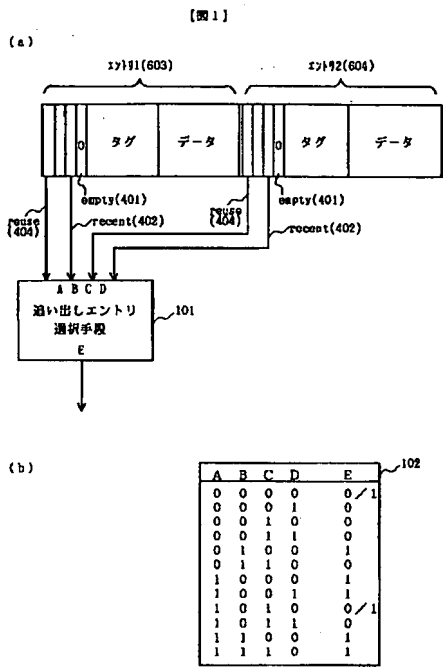
【図９】従来のキャッシュの動作を説明するために図８のプログラムを機械語イメージに近づけたプログラムを示す図である。

【図１０】本発明のキャッシュの動作を説明するために図８のプログラムを機械語イメージに近づけたプログラムを示す図である。

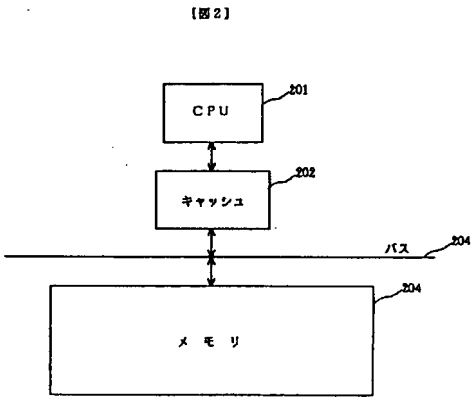
【符号の説明】

１０１　追い出しエントリ選択手段
２０１　ＣＰＵ
２０２　キャッシュ
２０３　バス
２０４　メモリ
３０１　セット１
３０２　セット２
３０３　制御ビット
３０４　タグ
３０５　データ
３０６　ライン
３０７　インデックス
４０１　ｅｍｐｔｙビット
４０２　ｒｅｃｅｎｔビット
４０３　ｄｉｒｔｙビット
４０４　ｒｅｕｓｅビット
６０３　エントリ１
６０４　エントリ２

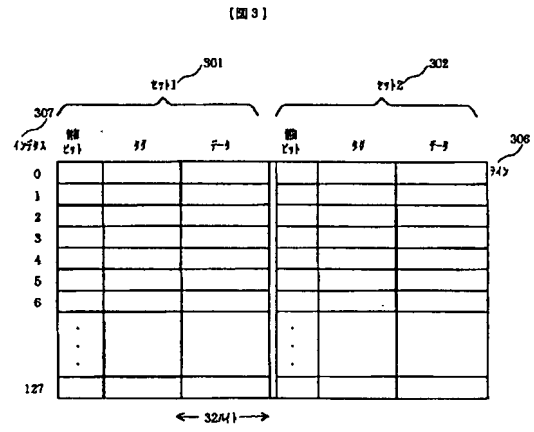
【図1】



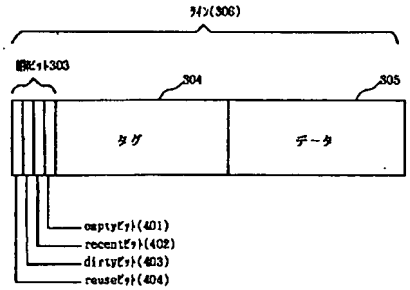
【図2】



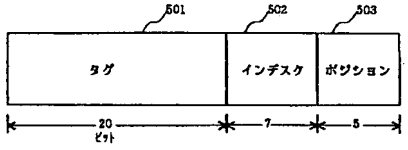
【図3】



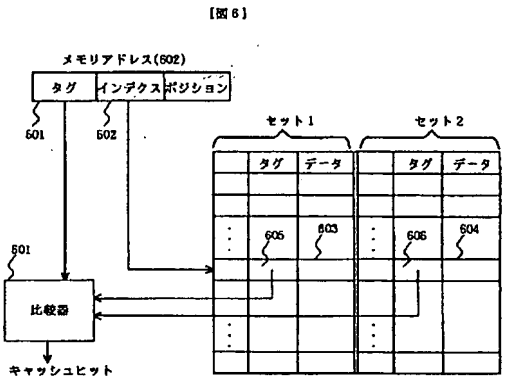
【図4】



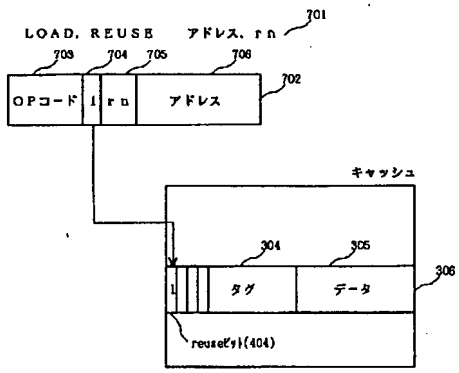
【図5】



【図6】



【図7】



【図8】

```
int s,a[1024][1024],b[1024];                    ~801
for (i=0; i<1024; i+=2)                         ~802
  for (j=0; j<1024; j++)                        ~803
    s=s+a[i][j]+a[i+1][j]+b[j];                 ~804
```

【図9】

```
int s,a[1024][1024],b[1024];                    ~901
for (i=0; i<1024; i+=2)                         ~902
  for (j=0; j<1024; j++){                       ~903
    load(r0, &a[i][j]);                         ~904
    load(r1, &a[i+1][j]);                       ~905
    load(r2, &b[j]);                            ~906
    s=s+r0+r1+r2;                               ~907
  }                                             ~908
```

【図10】

```
int s,a[1024][1024],b[1024];                    ~1001
for (i=0; i<1024; i+=2)                         ~1002
  for (j=0; j<1024; j++){                       ~1003
    load(r0, &a[i][j], NO_REUSE);               ~1004
    load(r1, &a[i+1][j], NO_REUSE);             ~1005
    load(r2, &b[j], REUSE);                     ~1006
    s=s+r0+r1+r2;                               ~1007
  }                                             ~1008
```